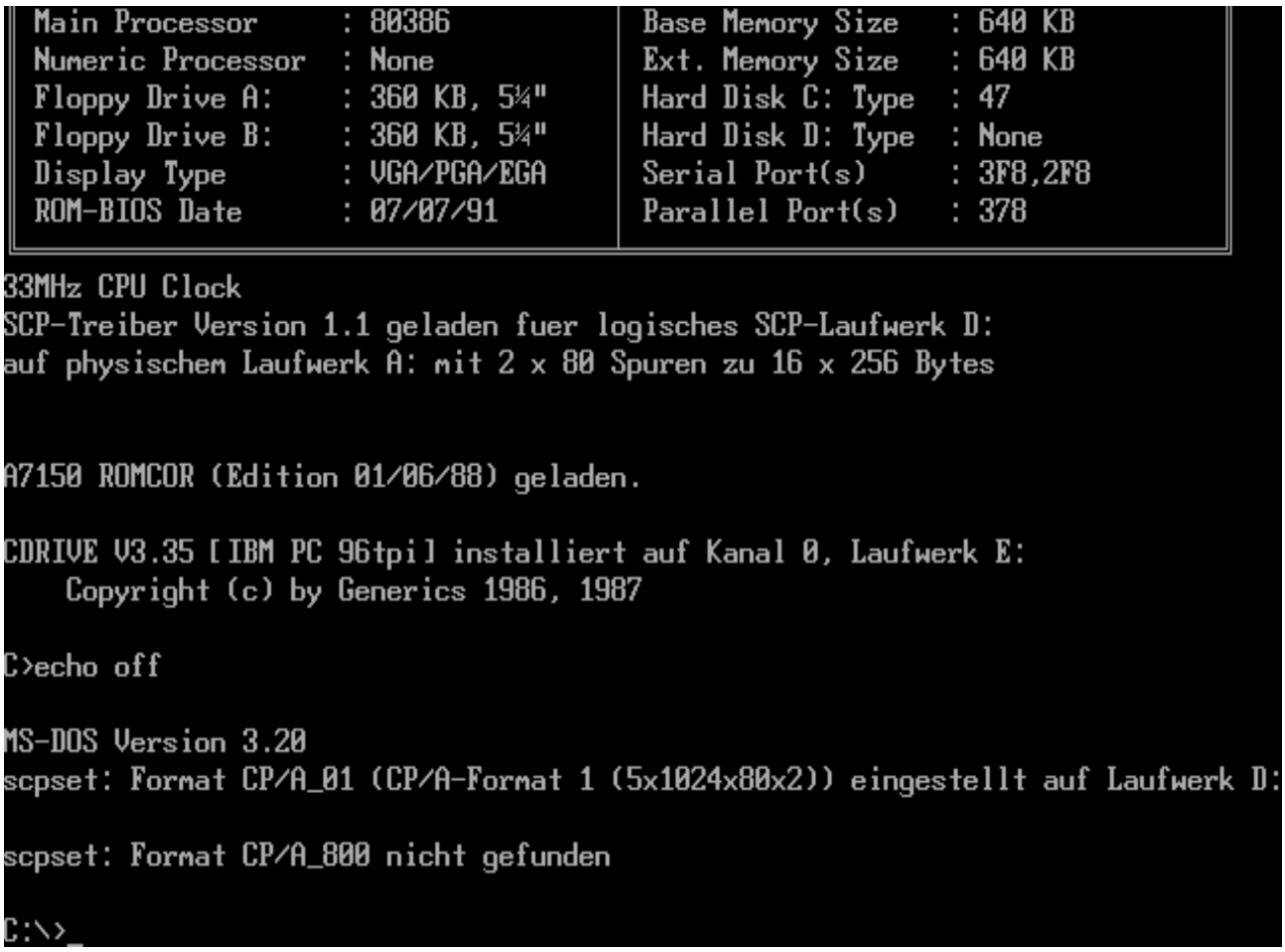


Beschreibeung IMG_TEST --- SCP bzw. CP/M Diskettenformat-Test.

IMG_TEST versucht der Reihe nach alle Profilnamen anzuwenden, bis mit
einem Profilname der Zugriff auf das gewählte logische Laufwerk funktioniert.
Dann schreibt es den Profil-Namen aus und beendet sich.
Das Programm IMG_TEST läuft unter DOS 3.2.

Es benutzt die installierten DOS-Treiber SCPDR.SYS oder CDRIVE.SYS (von SCOPY),
um auf das physische Diskettenlaufwerk A: über logische
Laufwerksbezeichner D: oder E: zuzugreifen (B: funktioniert nicht).
Die DOS-Treiber dürfen max. zweimal installiert werden für zwei logische Laufwerke.
IMG_TEST kann nur mit den logischen Laufwerken D: und E: arbeiten.
Es müssen DEUSCHSPRACHIGE Versionen sein, da mangels Rückgabecode der
Test auf die Bildschirmausgabe des Wortes "nicht" erfolgt ! (Anpassung im Quelltext möglich)

IMG_TEST wird durch PROFILE.DAT gesteuert. In dieser Datei sind die Namen der
zu testenden Formate gespeichert, je Zeile ein Profile-Name.
Diese Namen sollten natürlich auch in SCPSET.DAT (bei Benutzung von SYPDR.SYS) oder
SELECT.DAT (bei Benutzung von CDRIVE.SYS) enthalten sein.

Es kann natürlich sein, das dieses Profil zwar formal funktioniert, aber keine gültigen Ergebnisse liefert. Dann kann man es mit Semikolon in PROFILE.DAT auskommentieren und die Prüfung erneut starten.

Alle Dateien müßen im AKTUELLEN Pfad liegen !
Der Aufruf erfolgt in der Form: IMG_TEST {s | c} {d:| e:}
Beide Parameter müssen angegeben werden !
's' benutzt CDRIVE.SYS und 'c' benutzt SCPDR.SYS.

IMG_TEST.EXE wird wie folgt Kompiliert:

```
DEL BTEST.OBJ
DEL ERROR.TXT
DEL BTEST.EXE
masm  BTEST.ASM  BTEST.OBJ  ERROR.TXT, NUL;
link BTEST.OBJ;
```

Benutzungsvoraussetzung ist die installatioen eines oder maximal zweier logischer Laufwerke mittels der Treiber:

```
device=c:\DCP_320\scpdr.sys /D:0
device = \scopy\cdrive.sys -r2 -u0 -g
```

Die Namen der logischen Laufwerke werden durch die Treiberreihenfolge bestimmt, hier D: und E:

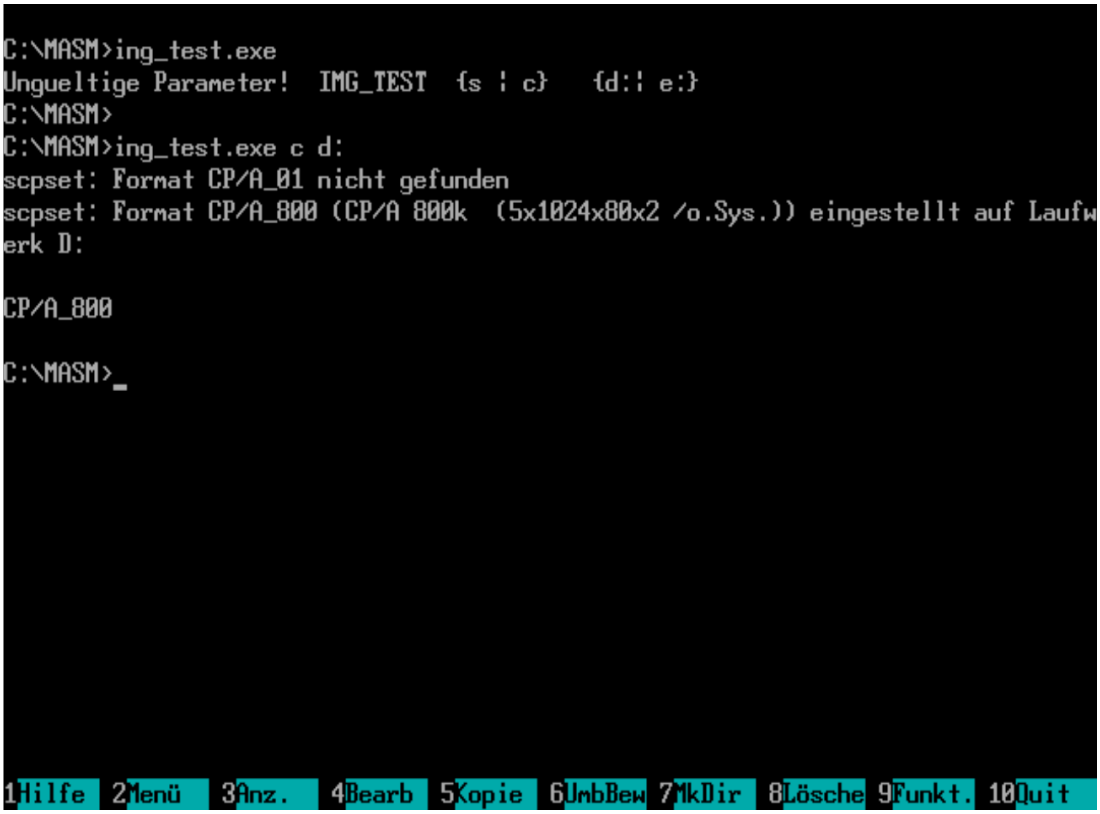
PROFILE.DAT könnte folgenden Inhalt haben:

```
CP/A_01
CP/A_800
SCPF_03
SCPF_04
SCPF_05
```

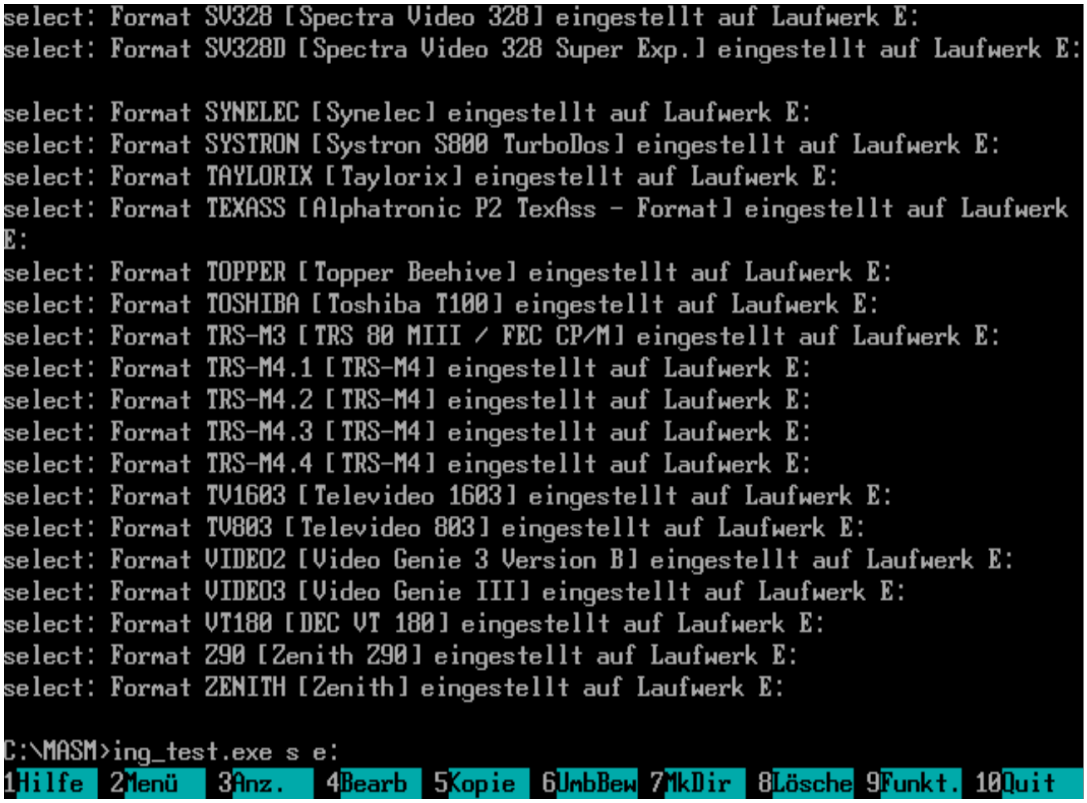
Benötigt werden die Programme SCPSET.COM und / oder SELECT.EXE sowie die Formatdefinitionsdatei SELECT.DAT.

Das Programm läuft auch in 86Box.

Eine Bildschirmausgabe mit SCPDR.SYS (erfolgreich) könnte wie folgt aussehen:



Bildschirmausgabe 2 mit CRIVE.SYS zeigt eine nicht erfolgreiche Prüfung:



```
;  Das Programm IMG_TEST läuft unter DOS 3.2
;
;  Es benutzt die installierten DOS-Treiber  SCPDR.SYS  oder CDRIVE.SYS (von SCOPY),
;  um auf das physische Diskettenlaufwerk A: über logische
;  Laufwerksbezeichner D: oder E: zuzugreifen (B: funktioniert nicht).
;  Die DOS-Treiber dürfen max. zweimal installiert werden für zwei logische Laufwerke.
;  IMG_TEST kann nur mit den logischen Laufwerken D: und E: arbeiten.
;  Es müssen DEUSCHSPRACHIGE Versionen sein, da mangels Rückgabecode der
;  Test auf die Bildschirmausgabe des Wortes "nicht" erfolgt ! (Anpassung im Quelltext möglich)
;
;  IMG_TEST wird durch PROFILE.DAT gesteuert. In dieser Datei sind die Namen der
;  zu testenden Formate gespeichert, je Zeile ein Profile-Name.
;  Diese Namen sollten natürlich auch in SCPSET.DAT (bei Benutzung von SYPDR.SYS) oder
;  SELECT.DAT (bei Benutzung von CDRIVE.SYS) enthalten sein.
;  IMG_TEST versucht der Reihe nach alle Profilnamen anzuwenden, bis mit
;  einem Profilname der Zugriff auf das gewählte logische Laufwerk funktioniert.
;  Dann schreibt es den Profil-Namen aus und beendet sich.
;
;  Alle Dateien müßen im AKTUELLEN Pfad liegen !
;  Der Aufruf erfolgt in der Form:  IMG_TEST  {s | c}   {d:| e:}
;  Beide Parameter müssen angegeben werden !
;  's'  benutzt CDRIVE.SYS und  'c'  benutzt SCPDR.SYS.
;

STACKSEG SEGMENT STACK
   DB 64 DUP(?)
STACKSEG ENDS

DATASEG SEGMENT
   filename          DB 'PROFILE.DAT',0
   buffer            DB 128 DUP(0)
   oldHandler        DW 0,0,0,0
   fileHandle        DW 0
   out_line          DB 80 DUP(?)      ; Zielpuffer für gelesene Zeichen
   program_name      DB 20 dup(0)
   program_name1     DB 'SCPSET.COM',0
   program_name2     DB 'SELECT.EXE',0
   parameter_block   DW 0              ; Umgebung übernehmen
                     DD 0              ; Keine Kommandozeile -- wird überschrieben
                     DD 0              ; Kein FCB1
                     DD 0              ; Kein FCB2
   vgl_ret           DB 0
   test_filename     DB 'D:\*.*',0
   p_prog_typ        DB 0
   p_lw_typ          DB 0
   p_fehlertext      DB 'Ungueltige Parameter!  IMG_TEST  {s | c}   {d:| e:}$'
   scp_error_msg     DB 'SCPSET fehlt im aktuellen Pfad.$'
   scopy_error_msg   DB 'SELECT fehlt im aktuellen Pfad.$'
   profile_error_msg DB 'PROFILE.DAT fehlt im aktuellen Pfad.$'

DATASEG ENDS
```

```
CODESEG SEGMENT
ASSUME CS:CODESEG, DS:DATASEG, SS:STACKSEG

ErrorHandler PROC FAR
   MOV AL, 3
   IRET
ErrorHandler ENDP

; ---------------------------------------

parameterpruefung PROC
   ; ES muss auf PSP zeigen (Segment 0x0000)
   ; Parameterzeile aus PSP lesen und ausgeben
   MOV    AH, 62h          ; DOS-Funktion: Get PSP segment
   INT    21h
   MOV    ES, BX           ; ES zeigt jetzt auf PSP → Anfang des Speicherblocks

   ; Parameterlänge aus PSP:80h holen
   MOV CL, ES:[80h]   ; Command-Line-Length
   TEST CL, CL
   JNZ p_weiter
   JMP p_fehler     ; JMP kann NEAR oder FAR sein
p_weiter:

   ; SI = Adresse PSP:81h Start Parameterzeile
   MOV BX, 81h  ; BX zeigt auf erstes Zeichen der Kommandozeile
   ; DI = Zielpufferadresse (p_puffer)
   ; CX Zeichen kopieren

   ; Erstes Nicht-Leerzeichen suchen
find_first:
   MOV AL, ES:[BX]
   CMP AL, ' '
   JNE check_first
   INC BX
   LOOP find_first
   JMP p_fehler     ; Nur Leerzeichen → Fehler

check_first:
   CMP AL, 's'
   JE set_prog_s
   CMP AL, 'S'
   JE set_prog_s
   CMP AL, 'c'
   JE set_prog_c
   CMP AL, 'C'
   JE set_prog_c
   JMP p_fehler

set_prog_s:
   MOV p_prog_typ, 2    ; scopy
   JMP skip_spaces
```

```
set_prog_c:
   MOV p_prog_typ, 1    ; scpset
   JMP skip_spaces

skip_spaces:
   INC BX               ; Nächstes Zeichen nach erstem Parameter

skip_loop:
   CMP CX, 0
   JE p_fehler
   MOV AL, ES:[BX]
   CMP AL, ' '
   JNE check_second
   INC BX
   DEC CX
   JMP skip_loop

check_second:
   CMP AL, 'd'
   JE check_colon_d
   CMP AL, 'D'
   JE check_colon_d
   CMP AL, 'e'
   JE check_colon_e
   CMP AL, 'E'
   JE check_colon_e
   JMP p_fehler

check_colon_d:
   INC BX
   CMP CX, 0
   JE p_fehler
   MOV AL, ES:[BX]
   CMP AL, ':'
   JNE p_fehler
   MOV p_lw_typ, 'D'
   MOV BYTE PTR test_filename[0], 'D'
   RET

check_colon_e:
   INC BX
   CMP CX, 0
   JE p_fehler
   MOV AL, ES:[BX]
   CMP AL, ':'
   JNE p_fehler
   MOV p_lw_typ, 'E'
   MOV BYTE PTR test_filename[0], 'E'
   RET
```

```
p_fehler:
   ; Fehler ausgeben
   MOV AH, 09h
   LEA DX, p_fehlertext
   INT 21h
   MOV AX, 4C01h
   INT 21h

p_ende:
   RET
parameterpruefung ENDP

; ---------------------------------------

PrintBuffer PROC NEAR
   PUSH SI
   push DX
   push AX

   MOV SI, OFFSET BYTE PTR buffer[1]
   MOV AH, 0Eh

pl:
   LODSB
   CMP AL, 0
   JE pd
   INT 10h
   JMP pl
pd:
   call PrintWrzl

lret:
   pop AX
   pop DX
   POP SI

   RET

PrintBuffer ENDP

; ---------------------------------------

PrintName PROC NEAR
   PUSH SI
   push DX
   push AX

   MOV SI, OFFSET BYTE PTR program_name[0]
   MOV AH, 0Eh

pl2:
   LODSB
```

```
    CMP AL, 0
    JE pd2
    INT 10h
    JMP pl2
pd2:
    call PrintWrzl

lret2:
    pop AX
    pop DX
    POP SI

    RET

PrintName ENDP

; ---------------------------------------
; ---------------------------------------

MAIN PROC

    ; Initialisiere DS
    MOV AX, DATASEG
    MOV DS, AX
    MOV ES, AX

    CALL    parameterpruefung
    CALL    test_profile_dat
    CALL    FindProgramInPath

; INT 24h sichern
    MOV    AH, 35h
    MOV    AL, 24h
    INT    21h
    MOV    WORD PTR oldHandler, BX
    MOV    WORD PTR oldHandler+2, ES

; INT 24h setzen
    MOV    DX, OFFSET ErrorHandler
    MOV    AX, SEG ErrorHandler
    MOV    DS, AX
    MOV    AH, 25h
    MOV    AL, 24h
    INT    21h

; DS zurücksetzen
    MOV    AX, DATASEG
    MOV    DS, AX

; Datei öffnen
    MOV    AH, 3Dh
    MOV    AL, 0
```

```
    MOV     DX, OFFSET filename
    INT     21h
    JC      exit

    MOV     fileHandle, AX   ; Wenn kein Fehler → Handle speichern
    call    ReadAndPrintLines

exit:
    MOV AH, 3Eh
    MOV BX, fileHandle
    INT 21h

    ; INT 24h wiederherstellen
    MOV DX, WORD PTR oldHandler
    MOV DS, WORD PTR oldHandler+2
    MOV AH, 25h
    MOV AL, 24h
    INT 21h

    MOV AH, 4Ch
    MOV AL, 0
    INT 21h
MAIN ENDP

; ------------------------------------------

ReadAndPrintLines PROC NEAR
    MOV AX, DATASEG
    MOV DS, AX
    MOV ES, AX

read_loop:
    XOR     DI, DI              ; Pufferindex zurücksetzen
    INC     DI
    XOR     SI, SI              ; Zähler für gelesene Bytes pro Zeile

read_char:
    MOV BX, fileHandle          ; Handle der geöffneten Datei
    LEA DX, buffer[DI]          ; Zieladresse des aktuellen Buffer-Schreibbyte laden
    MOV CX, 1                   ; Ein Byte lesen
    MOV AH, 3Fh                 ; DOS: Datei lesen
    INT 21h                   ; An Adresse <DX> speichern beginnen buffer[1]
    JC read_done                ; Fehler → Ende

    CMP AX, 0
    JE read_done                ; Dateiende → Ende

    MOV AL, buffer[DI]          ; Gelesenes Zeichen prüfen
    CMP AL, 0Dh                 ; Zeilenende (CR)?
    JE line_end

    CMP AL, 0Ah                 ; von vorhergehenden Zeilenende, 0ah wird verworfen
```

```
    JE read_char

    INC SI                  ; Zähler erhöhen
    INC DI                  ; Nächstes Zeichen
    CMP DI, 127             ; Puffergrenze?
    JB read_char            ; Noch Platz → weiter lesen

line_end:
    CMP DI, 1
    JB read_done            ; Leerzeile ->  Ende

    MOV BYTE PTR buffer[DI], 0    ; Nullterminierung
    DEC DI
    MOV AX, DI
    MOV BYTE PTR buffer[0], al    ; Längenbyte

    CALL set_profile        ; Verarbeitung der Zeile
    CALL lies_bildtext

    ; Prüfe, ob die gelesene Zeile mit "nicht" beginnt
    ; Wenn ja → vgl_ret = 1, sonst = 0
    ; Wenn vgl_ret == 0 → Programm beenden
    CMP BYTE PTR vgl_ret, 0
    JNE read_loop

    MOV AH, 4Eh             ; Find First File
    MOV CX, 0               ; Normaler Attributfilter
    MOV DX, offset test_filename  ; ASCIIZ-String "D:\*.*"
    INT 21h
    JC  read_loop           ; Carry gesetzt → Fehler (z. B. kein Datenträger)

    ; Erfolg → Laufwerk ist lesbar
    CALL   Printwrzl
    CALL   PrintBuffer

read_done:
    RET
ReadAndPrintLines ENDP


; ---------------------------------

PrintWrzl PROC
    push ax
    mov ah, 0Eh
    mov al, 13
    int 10h
    mov al, 10
    int 10h
    pop ax
    ret
PrintWrzl ENDP
```

```
; ---------------------------------

set_profile PROC

   push   ax
   push   bx
   push   cx
   push   dx
   push   si
   push   di
   push   bp
   push   ds
   push   es

   MOV    AH, 62h          ; DOS-Funktion: Get PSP segment
   INT    21h
   MOV    ES, BX           ; ES zeigt jetzt auf PSP → Anfang des Speicherblocks

   ; Verkleinere den Speicherblock auf z. B. 32 KB
   MOV    AH, 4Ah          ; DOS-Funktion: Set Block
   MOV    BX, 512          ; Anzahl Paragraphen (512 × 16 = 8192 Bytes = 8 KB)
   INT    21h
   JNC    weiter1      ; Wenn Carry gesetzt → Fehlerbehandlung

   ADD    AL, 30h          ; AL als ASCII-Zeichen (z. B. Fehlercode)
   MOV    AH, 0Eh          ; BIOS: TTY-Ausgabe
   MOV    BH, 0            ; Seite 0
   MOV    BL, 3            ; Textfarbe (hellgrau)
   INT    10h              ; Zeichen ausgeben
   INT    10h              ; Zeichen ausgeben
   mov    al,'+'
   INT    10h              ; Zeichen ausgeben
   jmp    weiter

weiter1:
   MOV    AX, DATASEG
   MOV    DS, AX
   MOV    ES, AX

   mov    dx, offset program_name
   mov    bx, offset parameter_block
   mov    word ptr parameter_block[02h], offset buffer
   mov    word ptr parameter_block[04h], es
   mov    word ptr parameter_block[06h], 5Ch
   mov    word ptr parameter_block[08h], es
   mov    word ptr parameter_block[0Ah], 6Ch
   mov    word ptr parameter_block[0Ch], es
   mov    al,0
   mov    ah, 4Bh
   mov    al, 00h
   int    21h
```

```
    JNC     weiter       ; Wenn Carry gesetzt → Fehlerbehandlung

    ADD     AL, 30h      ; AL als ASCII-Zeichen (z. B. Fehlercode)
    MOV     AH, 0Eh      ; BIOS: TTY-Ausgabe
    MOV     BH, 0        ; Seite 0
    MOV     BL, 4        ; Textfarbe ROT
    INT     10h          ; Zeichen ausgeben
    mov     al,' '
    INT     10h          ; Zeichen ausgeben

weiter:
    pop     es
    pop     ds
    pop     bp
    pop     di
    pop     si
    pop     dx
    pop     cx
    pop     bx
    pop     ax

    ret

set_profile ENDP

; ----------------------------------

lies_bildtext PROC NEAR
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI
    PUSH DI
    PUSH ES

    MOV AX, 0B800h          ; Videospeichersegment
    MOV ES, AX

    ; Cursor steht auf Zeilenanfang → ermittle aktuelle Zeile
    ; BIOS-Funktion INT 10h / AH=03h → Cursorposition holen
    MOV AH, 03h
    MOV BH, 0               ; Seite 0
    INT 10h
    ; DH = Zeile, DL = Spalte

    DEC DH                  ; Vorherige Zeile
    MOV AL, DH
    CBW                     ; AL → AX (Zeilennummer)
    MOV BL, 80
    MUL BL                  ; AX = Zeile * 80
```

```
    ADD AX, 16
    MOV DL, BYTE PTR buffer[0]
    XOR DH, DH              ; DH = 0 → DX = 16-Bit-Wert
    ADD AX, DX              ; DX = AX + buffer[0]

    SHL AX, 1           ; *2 → Byte-Offset
    MOV SI, AX          ; SI = Startadresse im Videospeicher

    LEA DI, out_line    ; Zielpuffer
    MOV CX, 14          ; Anzahl Zeichen

read_loop8:
    MOV AL, ES:[SI]     ; Zeichen lesen
    MOV [DI], AL        ; In Puffer schreiben
    ADD SI, 2           ; Nächstes Zeichen (2 Bytes pro Zeichen)
    INC DI
    LOOP read_loop8

    ; Vergleich: out_line[0..4] == "nicht"
    LEA SI, out_line

    MOV AL, [SI]    ; 'n'
    CMP AL, 'n'
    JNE set_zero

    MOV AL, [SI+1]  ; 'i'
    CMP AL, 'i'
    JNE set_zero

    MOV AL, [SI+2]  ; 'c'
    CMP AL, 'c'
    JNE set_zero

    MOV AL, [SI+3]  ; 'h'
    CMP AL, 'h'
    JNE set_zero

    MOV AL, [SI+4]  ; 't'
    CMP AL, 't'
    JNE set_zero

    ; Alle Zeichen stimmen → setze vgl_ret = 1
    MOV BYTE PTR vgl_ret, 1
    JMP skip_check

set_zero:
    MOV BYTE PTR vgl_ret, 0

skip_check:


    POP ES
```

```
    POP DI
    POP SI
    POP DX
    POP CX
    POP BX
    POP AX
    RET
lies_bildtext ENDP

; ----------------------------------

FindProgramInPath PROC NEAR
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI
    PUSH DI
    PUSH DS
    PUSH ES

    MOV AX, DATASEG
    MOV DS, AX
    MOV ES, AX

    ; Prüfe p_prog_typ
    MOV AL, p_prog_typ
    CMP AL, 1
    JE check_scp
    CMP AL, 2
    JE check_scopy
    JMP exit_invalid_type     ; Ungültiger Typ

check_scp:
    LEA DX, program_name1
    MOV CX, 0
    MOV AH, 4Eh
    INT 21h
    JC exit_scp_error         ; Datei nicht gefunden

    ; Kopiere SCPSET.COM in program_name
    LEA SI, program_name1
    LEA DI, program_name
copy_name1:
    LODSB
    STOSB
    CMP AL, 0
    JNE copy_name1
    JMP done

check_scopy:
    LEA DX, program_name2
```

```
    MOV CX, 0
    MOV AH, 4Eh
    INT 21h
    JC exit_scopy_error       ; Datei nicht gefunden

    ; Kopiere SELECT.EXE in program_name
    LEA SI, program_name2
    LEA DI, program_name
copy_name2:
    LODSB
    STOSB
    CMP AL, 0
    JNE copy_name2
    JMP done

done:
    POP ES
    POP DS
    POP DI
    POP SI
    POP DX
    POP CX
    POP BX
    POP AX
    RET

exit_scp_error:
    MOV AH, 09h
    LEA DX, scp_error_msg
    INT 21h
    MOV AX, 4C00h
    INT 21h

exit_scopy_error:
    MOV AH, 09h
    LEA DX, scopy_error_msg
    INT 21h
    MOV AX, 4C00h
    INT 21h

exit_invalid_type:
    MOV AH, 09h
    LEA DX, p_fehlertext
    INT 21h
    MOV AX, 4C00h
    INT 21h
FindProgramInPath ENDP

; ---------------------------------

test_profile_dat PROC NEAR
    PUSH AX
```

```
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI
    PUSH DI
    PUSH DS
    PUSH ES

    ; Dateiname in DX setzen
    LEA DX, filename
    MOV CX, 0             ; Normaler Attributfilter
    MOV AH, 4Eh           ; DOS-Funktion: Find First File
    INT 21h
    JC profile_not_found   ; Wenn Carry gesetzt → Datei nicht gefunden

    ; Datei gefunden → zurück
    POP ES
    POP DS
    POP DI
    POP SI
    POP DX
    POP CX
    POP BX
    POP AX
    RET

profile_not_found:
    MOV AH, 09h
    LEA DX, profile_error_msg
    INT 21h
    MOV AX, 4C00h
    INT 21h
test_profile_dat ENDP

; --------------------------------

CODESEG ENDS
END MAIN
```